Class 1

Professional Program: Data Administration and Management

MANAGING DATA(BASES) USING SQL (NON-PROCEDURAL SQL, X401.9)
AGENDA

1. Overview of SQL
   1.1 Introduction to SQL
   1.2 The History of SQL
   1.3 History of SQL Standards

2. Overview of Relational Databases
   2.1 Review of Relational Databases
   2.2 Normalization Rules and Relationships

3. SQL Create Table
   3.1 Creating Tables
   3.2 The ALTER Table Statement
   3.3 The DROP Table Statement
   3.4 Primary / Foreign Key Constraints
Overview of SQL
1.1 INTRODUCTION TO SQL

- SQL stands for Structured Query Language (pronounced "ess-que-el" rather than “sequel”)
- Tool for organizing, managing, protecting, and retrieving data stored in a database system
- Wide acceptance among many Database Management Systems (DBMS) (because of early development and standardization)
- Highly portable across the different database platforms
- Non-procedural language (declarative language):
  - Specify only the “What” and not the “How”
- In contrast, a procedural language (imperative language):
  - Specify an explicit sequence of steps to follow to produce a result

Note: A database management system (DBMS) also contains a procedural environment to supplement the non-procedural SQL language. The most common object in this environment is the Stored Procedure. This course covers only the non-procedural SQL language.
1.1 INTRODUCTION TO SQL

- SQL is a command-based language, and SQL statements are comprised of many different clauses.
- SQL statement begins with a verb (i.e. SELECT) followed by clauses made up of keywords and parameters (i.e WHERE, FROM, INTO, etc.)
- Clauses:
  - may specify the data to be acted on
  - provide more detail about what the statement is supposed to do
  - are either required or optional
- Order of the different clauses or parts of a SQL statement is prescribed and important
- Spaces, line breaks, or indentation are ok

The SQL language is subdivided into several language elements, including:
- Clauses, which are constituent components of statements and queries. (In some cases, these are optional.)
- Expressions, which can produce either scalar values, or tables consisting of columns and rows of data.
- Predicates, which specify conditions that can be evaluated to SQL three-valued logic (3VL) (true/false/unknown) or Boolean truth values and are used to limit the effects of statements and queries, or to change program flow.
- Queries, which retrieve the data based on specific criteria. This is an important element of SQL.
- Statements, which may have a persistent effect on schemata and data, or may control transactions, program flow, connections, sessions, or diagnostics.
- SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
- Insignificant whitespace is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.
1.1 INTRODUCTION TO SQL

- **Data Query Language (DQL):**
  - DQL statements retrieve data from the database in many different ways
  - Also analyze data (count, summarize, minimum and maximum values)
  - No changes to data

- **Data Manipulation Language (DML):**
  - DML statements alter data in the database
  - Commit (save) or rollback (undo) the changes

- **Data Definition Language (DDL):**
  - Creates the data containers (structures that hold the data)
  - DDL language allows to Create, Drop, and Alter database objects

- **Data Control Language (DCL):**
  - Access to the database system (system privileges)
  - Access to the database objects as well as the data themselves (object privileges)
1.1 INTRODUCTION TO SQL

- Relational database model was introduced in 1970 by Dr. E.F. Codd
- SQL development started out here in Silicon Valley, at the IBM research center in Almaden
- System/R project was to prove the viability of the relational database model → first relational prototype mid 1970s
- New language developed: Structured English Query Language (SEQUEL)
- Second version in 1976, SEQUEL 2 → SQL
- Other vendors developed commercial products, such as Relational Software Inc. → 1979 first commercial DB (Oracle)
- UC Berkeley developed their first DB, called Ingres with its own language, Query Language (QUEL) → SQL
1.2 HISTORY OF SQL

- IBM finally brought their product to market in 1981 and called it SQL/Data system (SQL/DS). The second version of this product was called DB2, which is still around today.

- Sybase was yet another database vendor that developed quite successfully a solid database system called Sybase running under the UNIX operating system.

- Sybase entered into an agreement with Microsoft to develop the next version of Sybase (to be called System 10) with a version for Windows → SQL Server.

**Note:** Recently new types of databases evolved such as NoSQL. A NoSQL database provides a mechanism for storage and retrieval of data that uses looser consistency models than traditional relational databases. Motivations for this approach include simplicity of design, horizontal scaling and finer control over availability. NoSQL databases are often highly optimized key-value stores intended for simple retrieval and appending operations, with the goal being significant performance benefits in terms of latency and throughput. NoSQL databases are finding significant and growing industry use in big data and real-time web applications. NoSQL systems are also referred to as "Not only SQL" to emphasize that they do in fact allow SQL-like query languages to be used.

Current mainstream database systems include:

- Microsoft SQL Server, Microsoft Visual FoxPro, Microsoft Access
- Sybase
- MySQL (Oracle)
- Oracle
- IBM DB/2
- Postgre SQL
- Informix (IBM)
- Terradata
- SQLLite (part of C Library)
1.3 HISTORY OF SQL STANDARDS

- American National Standards Institute (ANSI) created a technical database committee named X3H2 (1982)
- First proposed standard was primarily based on IBM’s DB2 SQL dialect → first official SQL standard was ratified in 1986
- Adopted in 1987 by the International Organization of Standardization (ISO). Both standards refer to it as SQL/86
- Quickly criticized for lack of support for certain relational operators, lack of referential integrity, and redundancy with syntax → second version SQL/89
- These two standards were far from complete, but they formed an important foundation for the SQL language
1.3 HISTORY OF SQL STANDARDS

- New version of the SQL standard was adopted in 1992 by both organizations → SQL/92, also referred to SQL/2
- This standard was the first fairly complete standard and was much broader in scope than the previous standards
- Major milestone in developing a comprehensive SQL standard
- No database vendor currently fully supports and implements this standard
- Database vendors implemented their own versions due to market demands, additionally no ROI on some SQL features ($$)
- SQL/2 standard broken down into 3 parts:

**Entry SQL:**
This is the basic level that every major database product now supports. This level is easy to implement and should be followed by every database software.

**Intermediate SQL:**
This level encompasses the majority of the features of the SQL/92 standard. Up to today, most major database vendors are still trying to implement and integrate this level into their database products.

**Full SQL:**
Obviously, this is the complete SQL/92 standard.
1.3 History of SQL Standards

- Many database vendors implement their own flavor of SQL and add their own extension to SQL to overcome some major shortcomings of SQL.
- Adding vendor-specific extensions to SQL naturally makes the SQL language not portable anymore, one of the main goals of the SQL standard.
- Last major revision is SQL 99 standard, also referred to as SQL 3.

<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>Alias</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1986</td>
<td>SQL-86</td>
<td>SQL-87</td>
<td>First formalized by ANSI.</td>
</tr>
<tr>
<td>1999</td>
<td>SQL1999</td>
<td>SQL3</td>
<td>Added regular expression matching, recursive queries, triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features.</td>
</tr>
<tr>
<td>2006</td>
<td>SQL:2006</td>
<td>SQL 2006</td>
<td>Defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it enables applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.</td>
</tr>
</tbody>
</table>
Overview of Relational Databases
2.1 REVIEW OF RELATIONAL DATABASES

- Database is a collection of related data as a single unit
- A database possesses the following characteristics:
  - Management by a DBMS
  - Layers of data abstraction
  - Physical data independence
  - Logical data independence
2.1 REVIEW OF RELATIONAL DATABASES

- Database Models:
  - Flat Files
  - Hierarchical Model
  - Network Model
  - Relational Model
  - Object-Oriented (OO) Model
  - Object-Relational (OR) Model

- Databases classified by transactions:
  - On-line Transaction Processing (OLTP):
    - Data is highly fluid, many transactions
    - Uses minimum number of indexes, optimized for performance
  - Off-line Analytical Processing (OLAP):
    - Data is static, used in Data Warehouse, Data Mart, Data Mining
    - Many indexes to optimize searches and joins
2.2 Normalization Rules and Relationships

- Normalization is a process that results in a set of relations (tables) that possesses a unique set of properties.
- This process was developed by Dr. E.F. Codd in 1972 using three normal forms.
- Data Modelling → logical
- Developing DB → physical
- Every relation has a unique identifier.

Reorganize relations by applying normal forms:
- All non-key attributes fully depend on unique identifier.
- No other dependencies among non-key attributes.

<table>
<thead>
<tr>
<th>Logical Term</th>
<th>Physical Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation</td>
<td>Table</td>
</tr>
<tr>
<td>Unique Identifier</td>
<td>Primary Key</td>
</tr>
<tr>
<td>Attribute</td>
<td>Column, Field</td>
</tr>
<tr>
<td>Tuple</td>
<td>Row</td>
</tr>
</tbody>
</table>
A relation which fully complies with the normalization rules is said to be normalized.

In contrast, a relation that does not is denormalized.

A denormalized relation presents certain problems:

- **Insert Anomaly**
  - New tuple cannot be added because certain attributes cannot be inserted without the presence of other attributes (i.e. new course cannot be added without student).

- **Update Anomaly**
  - Update of a single value requires multiple tuples of data to be updated (i.e. update course title for course ID 23).

- **Delete Anomaly**
  - Deletion of a tuple causes unintended loss of data (i.e. deleting a student also deletes student enrollment).

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Course ID</th>
<th>Student Name</th>
<th>Course Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>23</td>
<td>Bill</td>
<td>Intro to relational DB</td>
</tr>
<tr>
<td>115</td>
<td>56</td>
<td>Mary</td>
<td>Advanced VB</td>
</tr>
<tr>
<td>113</td>
<td>23</td>
<td>Pam</td>
<td>Intro to relational DB</td>
</tr>
</tbody>
</table>
First Normal Form (1NF)

- Invoice contains lots of data
- Simply copy and paste into a spreadsheet is not beneficial or useful
- Take a closer look at the data:
  - Determine data relationships
  - Organize data into different entities
2.2 Normalization Rules and Relationships

- Invoice will require a few tables in a relational db
- First, put all data into one relation
- A unique identifier is a collection of one or more attributes that uniquely identifies each occurrence of a relation
- CustomerID uniquely identifies a customer, but a customer may have more than one invoice → not adequate for invoice
- Concatenate multiple attributes to achieve uniqueness
- If too many attributes → artificial or surrogate identifier
- CustomerID, OrderDate → limit to single order per day!!
- Create OrderID (artificial identifier)
- Sometimes more than one possible unique identifier:

<table>
<thead>
<tr>
<th>CustomerID</th>
<th>Customer Name</th>
<th>Order ID</th>
<th>Order Date</th>
<th>Product ID</th>
<th>Product Name</th>
<th>Quantity</th>
<th>Unit Price</th>
<th>Discount</th>
<th>Extended Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>BONAP</td>
<td>Bon App</td>
<td>11076</td>
<td>6-May-98</td>
<td>6</td>
<td>Grandma's Tofy Teatime</td>
<td>20</td>
<td>$25.00</td>
<td>25%</td>
<td>$375.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6-May-98</td>
<td>14</td>
<td></td>
<td>20</td>
<td>$23.25</td>
<td>25%</td>
<td>$346.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6-May-98</td>
<td>19</td>
<td></td>
<td>10</td>
<td>$5.20</td>
<td>25%</td>
<td>$69.00</td>
</tr>
</tbody>
</table>

Subtotal $792.75
Freight $38.29
Total $831.03
2.2 NORMALIZATION RULES AND RELATIONSHIPS

- Invoice Relation:
- 1NF:

A relation is said to be in first normal form when it contains no multi-valued attributes

Every intersection of a row and column must contain at most one value and no more

Invoice example: Several attributes are multi-valued and closely related, that is their values repeat together ➔ repeating group

1NF is about multi-valued attributes and repeating groups, and it takes more work to transform a relation into the 1NF. By convention, we enclose repeating groups in parentheses

```plaintext
# OrderID, CustomerID, Customer Name, Address..., Order Date, ProductID, Product Name, Quantity, Unit Price, Discount, Extended Price, Subtotal, Freight, Total
```

**Note:** The `#` symbol signifies the unique identifier.

```plaintext
# OrderID, CustomerID, Customer Name, Address..., Order Date, (ProductID, Product Name, Quantity, Unit Price, Discount, Extended Price), Subtotal, Freight, Total
```
To transform this relation into the first normal form, we must move all multi-valued attributes to another relation:

- Repeating groups must be moved together into same relation
- Individual multi-valued attributes should be moved to individual new relations rather than moved together into a new relation

New relations:

- Moved entire repeating group into new relation
- To avoid composite unique identifier, create LineItemNo

<table>
<thead>
<tr>
<th>Order relation:</th>
</tr>
</thead>
<tbody>
<tr>
<td># OrderID, CustomerID, Customer Name, Address..., Order Date, Subtotal, Freight, Total</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order Detail relation:</th>
</tr>
</thead>
<tbody>
<tr>
<td># Order ID, # ProductID, Product Name, Quantity, Unit Price, Discount, Extended Price</td>
</tr>
</tbody>
</table>

1. Create a new relation with an applicable name.
2. Copy the primary key from the original relation into the new relation where it becomes the foreign key.
3. Move the multi-valued attributes or repeating group to the new relation.
4. Create a primary key in the new relation by using the foreign key and additional attributes to make it unique in the new relation.
5. If too many attributes are needed, choose a surrogate key as a new primary key.
2.2 NORMALIZATION RULES AND RELATIONSHIPS

Second Normal Form (2NF)

- Main goal of the second normal form is to eliminate partial dependencies.

- Concept of functional dependency:
  - Attribute B is functionally dependent on attribute A if at any point in time there is only one value of attribute B for a given value of A.
  - A determines B, or A is a unique identifier of B.

A relation is said to be in second normal form under the following conditions:

- The relation is in first normal form.
- All non-key attributes are functionally dependent on the entire primary key.

- Second normal form only applies to relations with composite primary keys.
2.2 Normalization Rules and Relationships

- Apply the second normal form to the Order Detail relation.
- Primary key is comprised of OrderID and ProductID.
- Test dependencies of all non-key attributes to primary key.
- Non-key attribute: Product Name:
  - OrderID → Product Name: No, if we change OrderID, same product may be on the order.
  - ProductID → Product Name: Yes, if we change ProductID, then Product Name changes.
- Partial dependency violates 2NF!
- Move Product Name along with Unit Price and its primary key ProductID to a new relation named Product.

**Order relation:**

```
# OrderID, CustomerID, Customer Name, Address..., Order Date, Subtotal, Freight, Total
```

**Order Detail relation:**

```
# Order ID, # ProductID, Quantity, Discount, Extended Price
```

**Product relation:**

```
# ProductID, Product Name, Unit Price
```
2.2 NORMALIZATION RULES AND RELATIONSHIPS

Third Normal Form (3NF)

- Main goal of the third normal form is to eliminate transitive dependencies.
- Transitive dependency exists in a relation when one non-key attribute is functionally dependent on another non-key attribute.
- To summarize: Second and third normal form require that all non-key fields should only depend on the entire primary key and on nothing else in that relation.

Apply the third normal form:

- Move transitive dependent attributes to a relation where they depend only on the primary key.
- Leave the attribute on which it depends in the original relation as a foreign key.
2.2 NORMALIZATION RULES AND RELATIONSHIPS

- Extended Price attribute in the Order Detail relation:
  - Is a calculated attribute based on the product of unit price and quantity \( \rightarrow \) depends on another non-key attribute

- For any calculated attributes, the process is easy since we simply remove this attribute and do not place it anywhere

- Basic rules of relational databases not to store derived or calculated attributes

- Other non-key attributes that depend on each other: CustomerName and address information depend on CustomerID \( \rightarrow \) move this information into a new relation named Customer

- Invoice Example after normalizing

<table>
<thead>
<tr>
<th><strong>Customer relation:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td># CustomerID, CustomerName, Address....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Order relation:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td># OrderID, CustomerID, Order Date, Freight</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Order Detail relation:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td># Order ID, # ProductID, Quantity, Discount</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Product relation:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td># ProductID, Product Name, Unit Price</td>
</tr>
</tbody>
</table>
2.2 NORMALIZATION RULES AND RELATIONSHIPS

Relationships

- Association between two relations
- Lines between relations showing the minimum and maximum cardinality at both ends of each line
- The cardinality is the number of instances that one relation can be associated with the relation on the other side
- Maximum cardinality (one or many) is shown at the very end of the line
- Minimum cardinality (zero or one) just short before the end of the line
- Zero cardinality: o
- One cardinality: |
- Many cardinality: <

![Diagram showing relationships and cardinalities between Customers and Orders tables.](image-url)
One-To-One Relationship

- One instance in one relation can be associated with at most one instance of another relation, and vice versa.
- If the relationship is mandatory (instances in both relations must exist) → both tables can be merged into one table.
- If the relationship is not mandatory (one instance can exist without having an instance in the other relation) → a one-to-one relation may be warranted.
- Example: Employees and automobiles at car dealership
  - Employees entitled to a automobile based on seniority
  - Dealership has more automobiles than employees
  - Employees with no automobile assigned and automobiles with no employees assigned: Should be stored in two tables.
2.2 Normalization Rules and Relationships

One-To-Many Relationship

- One instance in one relation can be associated with zero to many instances in another relation, and one instance in the other relation can be associated with at most one instance in the first one.
- One-to-Many relationships are the most common relationship in a relation database, in fact, it is the very reason to avoid redundancy that occurs in a flat file table.
- It is rare for a one-to-many relationship to be optional on the one-side (parent), and even rarer to be mandatory on the many side (child).
Many-To-Many Relationship

- One instance in one relation can be associated with zero to many instances in another relation, and one instance in the other relation can be associated with zero to many instances in the first one.
- One Many-to-Many relationship between Orders and Products. One order may contain one or many products and one product may be contained on zero to many orders → new relation must be created.
- Order Details relation contain both keys from both many sides.
Recursive Relationships

- Relationships within one relation, that is, between different instances of that entity
- For example, an employee relation may contain information about who is the supervisor of an employee and who are the subordinates of that same employee
- Foreign key field must be contained in the same relation, Mgr_Emp_Id (foreign key) relating back to EmpID (primary key)

Identifying vs. Non-Identifying Relationships

- Solid relationship line represents an identifying relationship → foreign key is part of the primary key in the child table
- Dashed relationship line represents a non-identifying relationship → the foreign key in the child table is not part of the primary key
SQL Create Table
3.1 CREATING TABLES

- Table object is probably the most used and most important object.
- Syntax for managing table objects is quite complex due to the many properties of tables.

**Syntax:**

```
CREATE TABLE table_name
(
    column_name1  datatype(size) [DEFAULT expression],
    column_name2  datatype(size) [DEFAULT expression],
    column_name3  datatype(size) [DEFAULT expression]......
)
```

**Example 1-1:** Basic Create Table Statement

1. Issue the following CREATE TABLE statement creating a table named parent. Then use the Desc[ribe] command to view the basic properties of this table.

```sql
CREATE TABLE parent
(
    name  VARCHAR2(30),
    salary NUMBER(8,2) DEFAULT 0,
    bonus  NUMBER(6,2),
    email  VARCHAR2(50),
    date_created  DATE
);
```

**Warning:** You cannot issue the same CREATE TABLE command again since the table object already exists. You must first DROP the table before issuing the same command again.
3.2 ALTER TABLE STATEMENT

- Use the ALTER TABLE command to make changes to an existing table (no need to drop and recreate the table)
- ALTER TABLE command can be used on “live” tables in most situations
- Use the ALTER TABLE command for adding new columns, dropping columns, or modifying existing columns
- Add New Columns:

```sql
Syntax:
ALTER TABLE table_name
ADD
(column_name1 datatype(size) DEFAULT expression, column_name2 datatype(size) DEFAULT expression)
```

- Modify Existing Columns:

```sql
Syntax:
ALTER TABLE table_name
MODIFY
(column_name1 datatype(size) DEFAULT expression, column_name2 datatype(size) DEFAULT expression)
```
3.2 ALTER TABLE STATEMENT

- Rename Existing Columns:
  
  Syntax:
  
  ALTER TABLE table_name
  RENAME COLUMN column_old TO column_new

- Remove Existing Columns:
  
  Syntax:
  
  ALTER TABLE table_name
  DROP COLUMN column_name

Example 1-2: ALTER Table Statement: Adding New Column

1. Issue the ALTER TABLE statement to add another column named parent_id to table parent.
2. Then use the DESC[ribe] command to view the basic properties of this table.

```
SQL> ALTER TABLE parent
         2   ADD parent_id NUMBER(4,0);
Table altered.
SQL> DESCR parent
Name          | Null? | Type
---------------|-------|------
NAME           |       | VARCHAR2(30)
SALARY         |       | NUMBER(8,2)
BONUS          |       | NUMBER(6,2)
EMAIL          |       | VARCHAR2(50)
DATE_CREATED   |       | DATE
PARENT_ID      |       | NUMBER(4)
```
### 3.2 ALTER TABLE STATEMENT

#### Rules when altering tables

- You can increase the width or precision of a numeric column
- You can increase the width of a character column
- You can decrease the width of a column only if the column contains null values or the table has no rows at all
- You can change the data type only if the column contains null values or the table has no rows
- A change to the default value only affects subsequent new records

---

**Example 1-3: ALTER Table Statement: Modifying Existing Columns**

1. Issue the ALTER TABLE statement to modify the columns `parent_id` and `name`.
2. Then use the `Desc[ribe]` command to view the basic properties of this table.

```sql
SQL> ALTER TABLE parent
       MODIFY
          (       parent_id  NUMBER(6,0),
               name      VARCHAR2(50) );
Table altered.
SQL> DESC parent
Name | Null? | Type
-----|-------|-----
NAME |       | VARCHAR2(50)
SALARY |       | NUMBER(8,2)  
BONUS |       | NUMBER(6,2)  
EMAIL |       | VARCHAR2(50) 
DATE_CREATED |       | DATE
PARENT_ID |       | NUMBER(6)  
```
3.3 DROP TABLE STATEMENT

- Issue a DROP TABLE `table_name` command to delete a table and its data and indexes
- Database will not question this move, and if you have privileges, the table is irreversible removed
- Views and other database objects that depend on this table become invalid

**Syntax:**

```
DROP TABLE table_name;
```

**Example 1-4: DROP TABLE Statement**

1. Issue the DROP TABLE statement to remove the table parent from the database.
2. Then use the Desc[ribe] command to confirm that this table is removed.

```
SQL> DROP TABLE parent;
Table dropped.
SQL> DESC parent
ERROR: ORA-04043: object parent does not exist
```
3.4 PRIMARY/FOREIGN KEY CONSTRAINTS

- Five different constraints on a table object
- Most important one: Primary and Foreign Key constraint
- To enforce referential integrity, you must designate a primary key in the parent table first and then create a foreign key constraint on the child table that references the primary key of the parent table
- Enforcing referential integrity means:
  - You cannot delete a parent record when child records exist
  - Change the primary key value
  - Add child records referring to a primary key that does not exist (orphan records)
  - Change the foreign key value.
3.4 PRIMARY/FOREIGN KEY CONSTRAINT

- Define PRIMARY KEY constraint at the column level if the primary key consists of only one column or at the table level for a composite primary key
- One primary key constraint, but many foreign key constraints
- A FOREIGN KEY constraint establishes the referential integrity to a primary key in another table or even in the same table

**Syntax:**

```
CONSTRAINT foreign_key_constraint_name FOREIGN KEY(col1, col2, ...)
REFERENCES parent_table_name(col1, col2,..)
[ON DELETE CASCADE | ON DELETE SET NULL]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>foreign_key_constraint_name</td>
<td>Name of foreign key constraint</td>
</tr>
<tr>
<td>FOREIGN KEY</td>
<td>Foreign Key constraint type based on one or more columns</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>Name of parent table to be referenced</td>
</tr>
<tr>
<td>parent_table_name</td>
<td>Referencing one or more columns in the parent table</td>
</tr>
<tr>
<td>ON DELETE CASCADE</td>
<td>To delete child rows when corresponding parent rows are deleted.</td>
</tr>
<tr>
<td>ON DELETE SET NULL</td>
<td>To remove foreign key values and set to null when parent rows are deleted.</td>
</tr>
</tbody>
</table>
3.4 PRIMARY/FOREIGN KEY CONSTRAINT

**Example 1-6:** CREATE TABLE Statement including Primary Key/Foreign Key

1. Issue the following CREATE TABLE statement to create a child table having a foreign key reference to table parent.
2. Then use the Desc[ribe] command to view the basic properties of this table.

```sql
CREATE TABLE child
(
    child_id   NUMBER(4.0) CONSTRAINT child_id_pk PRIMARY KEY,
    parent_id   NUMBER(4.0),
    address1   VARCHAR2(30),
    address    VARCHAR2(30),
    city       VARCHAR2(30),
    state      VARCHAR2(2),
    zip_code   VARCHAR2(9),
    CONSTRAINT parent_id_fk FOREIGN KEY(parent_id) REFERENCES parent(parent_id)
);
Table created.
```

**Note:** The above example shows column level constraints (primary key) and table level constraints (foreign key). Table level constraints normally contain references to more than one column where column level constraints only apply to the column.